Name Table

What's in a name?

Wed, Sep 9, 1992

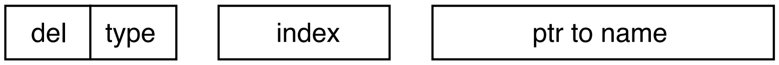
The VME local station systems have names that are used for analog channels. They may also have names for binary status bits. For a station with 2K analog channels, the time to execute the current linear search for a 6-character name can be 5 ms. This note describes an implementation of a generalized name table used for fast searching of names using a double hashing algorithm.

Initialization

At reset time a table is built that contains all the known names in the system. Each system table (in non-volatile memory) that houses names of anything is scanned, and the names are entered into a hashing table. Let each name length be even and up to 32 characters in length. The name is reduced to a longword and divided by the number of allocated table entries which is the *larger* prime number of a prime pair. In this way, the quotient is an index into the hashing table. If that entry is examined and found *not* empty, it is tested for a match against the search name. If there is a match, the name is already in the table. If there is no match, the next entry in sequence is examined, using as a sequencing delta a value obtained by dividing the same dividend by the *smaller* of the prime pair and adding 1. (This scheme is called double hashing, as it uses two hashing functions; it is described by Knuth in his volume on *Sorting and Searching*.) This matching procedure is continued in this way until an empty entry is found. Enter the new data into the empty entry. Only when the table is full will there be no empty entry found, although the efficiency of this scheme falls off somewhat before that happens.

Hash table entries

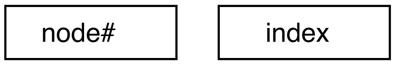
Suppose the information in the entry is 8 bytes as follows:



The `del` byte is a "deleted" flag. The `type` byte denotes the name type category, allowing for names of different types to match. The `index` word is the result data of a name lookup of a given type. (The most obvious example is an analog channel number.) The `ptr to name` is the address of a name field in a system table entry. To allocate room for 8K names, 64K bytes is required for the table. It should be made a bit larger to reduce the likelihood of collisions.

To process a request that specifies a name ident, the name table is consulted. If the hash code points to a non-empty entry, match against the name field pointed to. If there is no match, continue with the next entry until a match or an an empty entry is found. In the latter case, there is no match on the name, so the name does not exist. If a match is found, then the index word is returned to the requester.

The returned data has this format:



Insertions and deletions

To enter a new name into the table, follow the *Initialization* procedure described above. To delete an entry, the same sequence is followed, but upon finding an match, mark the entry deleted. It cannot be marked empty, since doing so may mask further entries in the chain.

```
Function NTLookup(VAR name: NameType; lng, nameType: Integer;
                  VAR index: Integer): Integer;

Function NTInsert(VAR name: NameType; lng, nameType: Integer;
                  index: Integer): Integer;

Function NTDelete(VAR name: NameType; lng, nameType: Integer): Integer;
```

Error return codes		*Routine*
0	No errors	All
–1	Invalid name table (All)	All
–2	#chars must be even and in range 2–32 (All)	All
–3	Name not in table	`NTLookup`, `NTDelete`
–4	Table overflow	`NTInsert`
–5	Duplicate name already in table	`NTInsert`
–6	Bus error. Bad ptr in table entry	All

Name changes

When a setting to an analog descriptor is made, a special check is made to determine whether the setting will result in changing the name field for that channel. If so, the current name is deleted and the new name added. This can cause table entries to be used up as entries are deleted, but they can be re-used. At reset time, rebuilding the table removes such "deleted" entries.

Name table header layout

NTKEY	NTSTRT	NTMSZ	NTVAC
NTREQS	NTCOLL	NTMAXC	NTLAST
NTFOUND	NTDELET	–	NTTIMZ
NTCNTR	NTDUPC	NTDUPL	NTTIME
NTCNT16	NTDCH16	NTDUP16	NTTIM16
–	–	–	–
–	–	–	–
–	–	–	NTTIMT

NTKEY	Name Table Key 'NT'
NTSTRT	Offset to first entry
NTMSZ	Maximum #entries in table
NTVAC	# Vacant entries in table
NTREQS	# times NameSrch called (diag)
NTCOLL	# collisions during NameSrch (diag)
NTMAXC	Max # collisions for a name (diag)
NTLAST	Last entry inserted (diagnostic)
NTFOUND	Offset to last name match
NTDELET	#entries deleted
–	spare(1)
NTTIMZ	Time to clear table during NTINTZ
NTCNTR	Count of 6-char names inserted
NTDUPC	Last duplicate 6-char name found
NTDUPL	Count of duplicate names
NTTIME	Time to enter all 6-char names (0.5 ms)
NTCNT16	Count of 16-char names inserted
NTDCH16	Last duplicate 16-char name found
NTDUP16	Count of duplicate 16-char names
NTTIM16	Time to enter 16-char names (0.5 ms)
–	spare(11)
NTTIMT	Total time to initialize name table